HPC for Graphs

Geometric and Combinatorial Algorithms for Load Balancing and Graph Mining

Henning Meyerhenke, Maria Predari, Charilaos Tzovas, Alexander van der Grinten, *Department of Computer Science, Humboldt-Universität zu Berlin*

In Short

- Graphs are widely used in the design and implementation of many methods in computer science, including numerical simulations. We research fast algorithms for two different categories of graph problems.
- We extend graph partitioning and process mapping tools that accelerate existing applications in scientific computing by providing better load balancing and by minimizing communication costs.
- We develop new parallel graph mining algorithms for HPC systems that extract knowledge from graphs arising in various application domains.

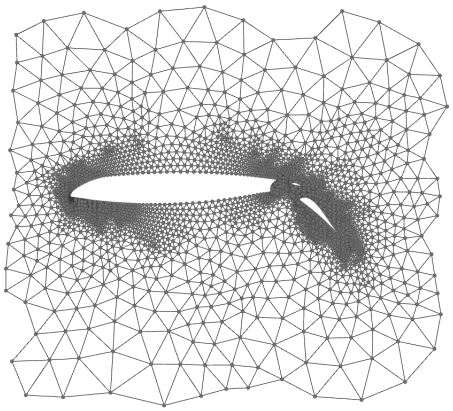

Figure 1: *Simulation mesh (i. e., communication graph) of an airfoil simulation.*

In computer science and mathematics, *graphs* (or corresponding sparse matrices) are a ubiquitous tool to express relationships between pairs of entities. Graphs arise both in mathematical codes (such as iterative solvers for sparse linear systems employed in numerical simulations, see Figure 1) and in domain-specific applications such as complex networks (e. g., computer networks or protein-protein interactions) or road networks. In this project, we develop and evaluate new graph algorithms for HPC clusters. Our objective is twofold: (i) we extend tools for *graph partitioning* and *process mapping* that accelerate existing HPC applications (e. g., numerical simulations that rely on sparse linear solvers) by improved load balancing capabilities and resource utilization on homogeneous and heterogeneous systems, and (ii) we develop new HPC-based *graph mining* tools that can be applied by domain scientists to extract domain-specific knowledge from large complex networks.

Graph Partitioning and Mapping

Solving large problem instances on HPC systems (e. g., solving sparse systems of linear equations in numerical simulations) often requires the division of the input into small parts such that each part can be distributed to a different processing unit (PU) such as a CPU or a GPU. The amount of communication between PUs depends on the quality of this distribution. Since communication between PUs is orders of magnitude slower than access to local data, minimizing communication is crucial for the efficiency of an HPC application.

A common strategy for computing a good distribution is to model the application's communication pattern as a graph. In the case of a numerical simulation, the communication graph often corresponds directly to the problem's simulation mesh (see Figure 1 for an example). On this graph, the graph partitioning problem is solved. The most common formulation of this problem asks for a division of the graph's vertex set V into k *blocks* (i. e., pairwise disjoint subsets) such that all blocks are no larger than $(1+\epsilon) \cdot \left\lceil \frac{|V|}{k} \right\rceil$ (for small $\epsilon \geq 0$), while some objective function modeling the communication volume is minimized. Afterwards, *process mapping* is applied, which maps each block of the partition to exactly one PU. Traditionally, the *edge cut* (i. e., the total number of graph edges having their incident vertices in different blocks) is used as a proxy for the communication volume. For a survey on the subject see [3].

Established combinatorial tools for general-purpose graph partitioning typically yield a high quality in terms of the edge cut. However, they do not scale to large numbers of PUs and require a large amount of extra memory (in addition to the memory required to store the input graph). Previous work often saw an increase in running time when moving beyond a few hundred PUs for such approaches [2]. Thus, for large-scale simulations the research community has moved to more scalable geometric methods, e. g., space-filling curves or re-

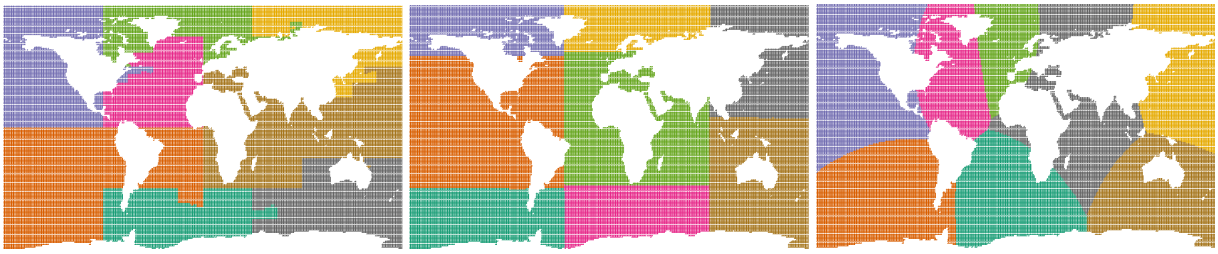

Figure 2: *Partitions of an ocean mesh by three different algorithms: space-filling curves, multi-section and our balanced k-means in Geographer. In this example, balanced k-means leads to small boundaries (i. e., low edge cut and communication volume) between blocks (= colored regions).*

cursive multi-section. While geometric methods are often much faster and scale better to large numbers of PUs than purely combinatorial ones, their partitioning quality is typically worse, leading to a higher running time of the targeted application.

In a previous BMBF project, we developed a balanced version of the k-means algorithm for direct k-way partitioning of geometric point sets corresponding to simulation meshes. Our algorithm and its implementation are called Geographer (*Geometry-based graph partitioner* [1]). Geographer scales to large input graphs and to large numbers of PUs. For example, in previous experiments on 16384 PUs, a mesh with 2 billion vertices can be partitioned into 16384 blocks within a few seconds. The partitions derived this way have good global shapes thus, resulting in a low edge cut compared to other geometric approaches (see Figure 2). In this project, we extend the balanced k-means approach of Geographer in multiple ways: (i) we want to support *repartitioning* of the graph at runtime to take changes in communication patterns into account, (ii) to be able to handle communication graphs of *heterogeneous* HPC systems, we want to support multiple balance constraints (e. g., to balance CPU and GPU load at the same time) and (iii) we want to integrate process mapping into the balanced k-means procedure instead of applying it in a postprocessing step.

Graph Mining

In graph mining, typical tasks ask to extract domain-specific knowledge (or information) from graphs. Exemplary graph mining tasks include identifying *central* (i. e., important) vertices of the graph, grouping similar vertices into clusters or predicting new relations between existing vertices. For a broad overview of graph mining applications, see Ref. [4]. Many graph mining algorithms are computationally expensive and require approximation and parallelism to be tackled in reasonable time. In the ongoing DFG project FINCA, we developed an approximation algorithm for the popular *betweenness centrality* measure that runs on HPC systems. In contrast to earlier codes, our implementation can often handle graphs with billions of edges in minutes on 384 PUs. In this project, we want to extend the scalability of our algorithm to larger numbers of PUs. Furthermore, our goal is to investigate *graph embedding* algorithms that embed graphs (which come without associated coordinates) into Euclidean spaces. This method can be used to make graphs accessible to other data mining and machine learning algorithms that assume the existance of feature vectors. Embeddings will also enable the use of Geographer's geometric techniques for graphs that do not *a priori* have associated coordinates.

WWW

`https://bit.ly/hub-macsy`

More Information

[1] M. von Looz, H. Meyerhenke, C. Tzovas: *Balanced k-means for Parallel Geometric Partitioning* (ICPP 2018).

[2] H. Meyerhenke, P. Sanders, C. Schulz: *Parallel Graph Partitioning for Complex Networks* (IEEE Trans. Parallel Distrib. Syst. 28(9), 2017).

[3] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, C. Schulz: *Recent Advances in Graph Partitioning* (Algorithm Engineering, 2016).

[4] M. Newman: *Networks* (Oxford University Press, 2018).

[5] A. van der Grinten, H. Meyerhenke: *Scaling Betweenness Approximation to Billions of Edges by MPI-based Adaptive Sampling* (preprint, 2019) `http://arxiv.org/abs/1910.11039`.

Funding

Fast Inexact Combinatorial and Algebraic Solvers for Massive Networks (FINCA) within DFG Priority Programme 1736